Estructuras Discretas

Práctica 07: Nuestras estructuras Pt.3

Azpeitia García Karyme Ivette.
Dorantes Perez Brando
Valencia Cruz Jonathan Josue

1/10/2021

1. Sea un árbol T (No necesariamente ordenado) demuesta usando inducción que:

```
nNodes(T) = length(inorder(T))
```

Demostración por Inducción sobre T

Caso Base Probamos que se cumple para el árbol '`Void`

Es fácil verificar, pues por como estan definidas las funciones `nNodes(Void), length([]),inorder(Void)` tenemos

```
nNodes(Void)=0=length([])=length(inorder(Void))
```

Hipótesis Inductiva

Supongamos que se cumple para el árbol T, esto quiere decir

`nNodes(T) = length(inorder(T))`

Paso Inductivo

Queremos ver que se cumple para un árbol más grande

P.d `nNodes(Node a T_i T_d) = length(inorder(Node a T_i T_d))`

Procedemos saliendo del lado derecho de la igualdad

```
length(inorder(Node a T_i T_d))
        = length(inorder(T_i)++[a]++inorder(T_d) ) --Por definición recursiva de inorder.
        = length(inorder(T_i))+length([a])+length(inorder(T_d)) --Por ser length distributiva.
        = nNodes(T_i)+1+nNodes(T_d) --Por Hipótesis Inductiva y def. de length.
        = 1 + nNodes(T_i) + nNodes(T__d) --Por conmutatividad.
        = nNodes(Node a T_i T_d)
```

Así que por principio de inducción se cumple para cualquier árbol T

2. El número de hijos que tenemos en nuestros árboles es a lo más dos, da una breve explicación de cómo podríamos hacerle para tener una cantidad ilimitada de hijos. Puedes agregar código, pero no es necesario.

```
data BTree a = Void | Node a ( BTree a ) ( BTree a ) deriving (Show , Eq)
```

Definimos la estructura de nuestros árboles con lo anterior, hay que notar que esto solo nos permite tener al menos dos hojas, dado que utiliza dos constructores `(BTree a)`, es decir si quisieramos tener árboles de al menos 3 hojas, se vería de la siguiente manera

```haskell
data BTree a = Void | Node a ( BTree a ) ( BTree a ) ( BTree a ) deriving (Show , Eq)
```

Y podriamos ir agregando cuantos constructores quisieramos para que el árbol tenga más hojas

```haskell
data BTree a = Void | Node a ( BTree a )_1 ( BTree a )_2 ( BTree a )_3 ...(BTree a)_n deriving (Show , E
```

Pero esto sería muy tardado y seguiríamos restringiendo la cantidad máxima, sin embargo sabemos que en haskell existen las listas, las cuales nos permiten tener una cantidad de datos ilimitados sin restringir en este caso la cantidad máxima de hojas, por lo que el data para este tipo de árboles sería el siguiente

```haskell
data BTree a = Void | Node a [BTree a ] deriving (Show , Eq)
```